

# $A^2T$ : Attend, Adapt and Transfer

## Attentive Deep Architecture for Adaptive Transfer from multiple sources

**Janarthanan Rajendran<sup>†</sup>**  
University of Michigan  
rjana@umich.edu

**Aravind Lakshminarayanan**  
Indian Institute of Technology, Madras  
aravindsrinivas@gmail.com

**Mitesh M. Khapra\***  
Indian Institute of Technology, Madras  
miteshk@cse.iitm.ac.in

**Prasanna P<sup>†</sup>**  
McGill University  
prasanna.p@cs.mcgill.ca

**Balaraman Ravindran**  
Indian Institute of Technology, Madras  
ravi@cse.iitm.ac.in

### Abstract

The ability to transfer knowledge from source tasks to a new target task can be very useful in speeding up a Reinforcement Learning agent. Such transfer has been receiving a lot of attention lately, yet the application of transfer poses two serious challenges which have not been adequately addressed. First, the agent should be able to avoid negative transfer, which happens when the transfer hampers or slows down the learning instead of helping it. Second, the agent should be able to do selective transfer, which is the ability to select and transfer from different and multiple source tasks for different parts of the state space of the target task. We propose  $A^2T$  (Attend, Adapt and Transfer), an attentive deep architecture for adaptive transfer, which addresses these challenges.  $A^2T$  is generic enough to effect transfer of either policies or value functions. Empirical evaluations on different learning algorithms show that  $A^2T$  is an effective architecture for transfer learning by being able to avoid negative transfer while transferring selectively from multiple sources.

### Introduction

One of the goals of Artificial Intelligence (AI) is to build autonomous agents that can learn and adapt to new environments. Reinforcement Learning (RL) is

a key technique for achieving such adaptability. The goal of RL algorithms is to learn an optimal policy for choosing actions that maximises some notion of long term performance. Transferring knowledge gained from tasks solved earlier to solve a new target task can help, either in terms of speeding up the learning process or in terms of achieving a better solution, among other performance measures. When applied to RL, transfer could be accomplished in many ways (see (Taylor and Stone, 2009; Taylor and Stone, 2011) for a very good survey of the field). One could use the value function from the source task as an initial estimate in the target task to cut down exploration (Sorg and Singh, 2009). Alternatively one could use policies from the source task(s) in the target task. This can take one of two forms - (i) the derived policies can be used as initial explorative trajectories (Atkeson and Schaal, 1997; Niekum et al., 2013) in the target task and (ii) the derived policy could be used to define *macro* actions which may then be used by the agent in solving the target task (Mannor et al., 2004; Brunskill and Li, 2014).

While transfer in RL has been much explored, there are two crucial issues that have not yet been dealt with satisfactorily. The first is *negative transfer* - when the transfer results in a performance worse compared to that of learning from scratch in the target task. This severely limits the applicability of transfer to cases in which some measure of relatedness between source and target tasks can be guaranteed. One work that explicitly addresses the question of negative transfer is that of (Brunskill and Li, 2014) where it is assumed that there is access to source tasks that sufficiently cover the space of

- 
- <sup>†</sup> Work done while authors were at IIT Madras
  - \* Work done while author was at IBM India Research Lab

problems from which the target task is drawn. Further, a safe exploration strategy is used to ensure that there is no negative transfer while learning the target task. In our work, we propose a more general approach. We maintain a network that learns from scratch on the target task (it uses the knowledge from source tasks when useful, indirectly). If there is evidence of negative transfer happening, the agent will fall back to this base network. The second problem with transfer is that of identifying an appropriate source task from which to transfer. In some scenarios, different source tasks might be relevant and useful for different parts of the state space in the target task. One way of approaching this, is to select different source tasks from which to transfer at different situations in the target task. We call this *selective transfer*. In our framework the agent can pick and use solutions from multiple and different source tasks while solving a target task, for different parts of its state space. This allows us to treat all the solutions of the prior task as a partial basis from which the target solution is created.

To this end, we propose  $A^2T$ : Attend, adapt and transfer, an attentive deep architecture for adaptive transfer, that avoids negative transfer while performing selective transfer from multiple source tasks. The key components of our architecture are a fail-safe solution (the base network that learns from random initialisation) that the agent can fall back to, on evidence of negative transfer; and a deep network that selects solutions of different source tasks as appropriate at different situations (state inputs). This selection is achieved by leveraging ideas from recent work on learning attention mechanisms through deep networks (Bahdanau et al., 2014), (Mnih et al., 2014), (Sorokin et al., 2015). We then demonstrate that this architecture is generic enough to effect transfer of either action policies or action-value functions, as the case may be. We also adapt different algorithms in reinforcement learning as appropriate for the different settings and empirically demonstrate that the  $A^2T$  is effective for transfer learning.

## Related Work

As mentioned earlier, transfer learning approaches could deal with transferring policies or value func-

tions. For example, (Banerjee and Stone, 2007) describe a method for transferring value functions by constructing a *Game tree*. Similarly, (Sorg and Singh, 2009) use the value function from a source task as the initial estimate of the value function in the target task.

Another method to achieve transfer is to reuse policies derived in the source task(s) in the target task. Probabilistic Policy Reuse as discussed in (Fernández and Veloso, 2006) maintains a library of policies and selects a policy based on a similarity metric, or a random policy, or a max-policy from the knowledge obtained. (Atkeson and Schaal, 1997; Niekum et al., 2013) propose a method to use the learned source policies as initial explorative trajectories in the target task instead of relying solely on random exploration. (Talvitie and Singh, 2007) try to find the promising policy from a set of candidate policies that are generated using different action mapping to a single solved task. In contrast, we make use of one or more source tasks to selectively transfer policies at the granularity of state. Apart from policy transfer and value transfer as discussed above, there is also some work (Ferguson and Mahadevan, 2006) on Representation transfer using Proto Value Functions.

We now mention a couple of works which address negative transfer and selective transfer. For example, (Lazaric and Restelli, 2011) address the issue of negative transfer in transferring samples for a related task in a multi-task setting. (Konidaris et al., 2012) discuss the idea of exploiting shared common features across related tasks. They learn a *shaping function* that can be used in later tasks.

Finally, we discuss two very recent works which are relevant. First, (Parisotto et al., 2015) explore transfer learning in RL across Atari games by trying to learn a multi-task network over the source tasks available and directly fine-tune the learned multi-task network on the target task. However, fine-tuning as a transfer paradigm cannot address the issue of negative transfer which they do observe in many of their experiments. (Rusu et al., 2016) try to address this issue by proposing a sequential learning mechanism where the filters of the network being learned for task  $k$  are dependent through lateral connections on the lower level filters of the networks learned already for the previous tasks. The idea is to

ensure that dependencies that characterize similarity across games could be learned through these lateral connections. Even though they do observe better transfer results than direct fine-tuning, they are still not able to *avoid* negative transfer in some of their experiments

In contrast, we introduce a separate attention network to learn how useful a source task is, relative to a network that just learns from random initialization as if there were no prior experts available. Adaptively through experiences generated, the architecture can assign low weight to an unfavorable source task if after sufficient exploration, it realizes that following a network learning from scratch is less harmful (more rewarding). Further, in contrast to previous works, our work explicitly focuses on the ability to *selectively transfer*, using multiple source tasks while avoiding *negative transfer*.

### Proposed Architecture, $A^2T$

Let there be  $N$  source tasks and let  $K_1, K_2, \dots, K_N$  be the solutions of these source tasks  $1, \dots, N$  respectively. Let  $K_T$  be the solution of the target task  $T$ . We propose a setting where  $K_T$  is learned as a function of  $K_1, \dots, K_N, K_R$ , where  $K_R$  is the solution learned from scratch while acting on the target task.  $K_R$  could use the solutions from the other source tasks indirectly to learn faster. In this work, we use a convex combination of the solutions of the source tasks to obtain  $K_T$ .

$$K_T(s) = w_{N+1,s}K_R(s) + \sum_{i=1}^N w_{i,s}K_i(s) \quad (1)$$

$$\sum_{i=1}^{N+1} w_{i,s} = 1, w_{i,s} \in [0, 1] \quad (2)$$

The agent uses  $K_T$  to act in the target task. Figure 1 shows the proposed architecture. The key component of the model is the central network which learns the weights ( $w_{i,s}$ ,  $i \in 1, 2, \dots, N+1$ ). We refer to this network as the *attention network*. The weights allow the network to selectively accept or reject the solutions of other source tasks depending on the input state. This ability allows the model to achieve both its stated goals, *viz.*, (i) avoid negative transfer from  $K_i(s)$  by setting  $w_{i,s}$  to a very low

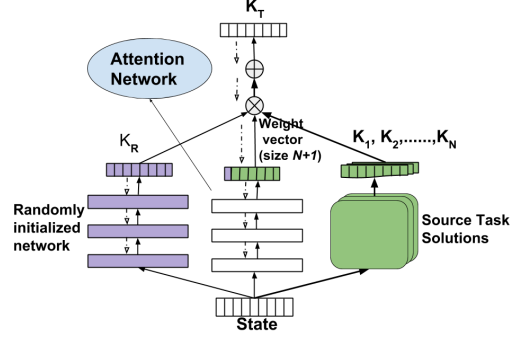


Figure 1:  $A^2T$  architecture for adaptive transfer. The dotted arrows represent the path of back propagation.

value if required and (ii) selectively transfer the solutions from multiple source tasks for certain states by assigning appropriate weights to those tasks specific to the states. We adopt a *soft-attention* mechanism whereby more than one weight can be non-zero (Bahdanau et al., 2014) as opposed a *hard-attention* mechanism (Mnih et al., 2014) where we are forced to have only one non-zero weight.

Depending on the feedback obtained from the environment upon following  $K_T$ , the attention network’s parameters are updated to improve performance. Even though the agent follows  $K_T$ , we update the parameters of the network that produces  $K_R$ , the randomly initialised network, as if the action taken by the agent was based only on  $K_R$ . The other source tasks solutions,  $K_1, \dots, K_N$  remain fixed. Updating these source tasks parameters would cause a significant amount of unlearning in the source tasks solutions and result in a weaker transfer, which we observed empirically. This also enables the use of source task solutions, as long as we have the outputs alone, irrespective of how and where they come from.

Due to the special way of updating  $K_R$ , apart from the unique contribution of  $K_R$  to  $K_T$ ,  $K_R$  results in replicating  $K_T$  at the end of learning. This also means that, if there is a source task whose solution  $K_j$  is useful for the target task in some parts of its state space, then over time,  $K_R$  would start replicating  $K_j$  in those parts of the state space. Note that the agent could follow/use  $K_j$  through  $K_T$  even before  $K_R$  attains its replication in the corresponding parts of the state space. Since the attention is soft,

our model has the flexibility to combine multiple source tasks. After the learning is done,  $K_R$  alone can be used to act in the target task for future endeavors and also gets added to our set of experts over various tasks. The use of deep neural networks allows the model to work even for large, complex RL problems. The deep attention network, allows the agent to learn complex selection functions, without worrying about representation issues a priori.  $A^2T$  is general and can be used for transfer of solutions such as policy and value.

### Policy Transfer

The solution that we transfer here is the source task's policy, taking advantage of which, we learn a policy for the target task. Thus, we have  $K_1, \dots, K_N, K_R \leftarrow \pi_1, \dots, \pi_N, \pi_R$ . When the attention network's weight for the policy  $\pi_R$  is high, the mixture policy is dominated by  $\pi_R$ , and the behavior is nearly on-policy. In the other case,  $\pi_R$  undergoes off-policy learning. Empirically, we observe that  $\pi_R$  converges. This architecture for policy transfer can be used alongside any algorithm that has an explicit representation of the policy. Here we describe two instantiations of  $A^2T$ , one for direct policy search using REINFORCE algorithm and another in the Actor-Critic setup.

### Policy Transfer in REINFORCE Algorithms using $A^2T$ :

REINFORCE algorithms (Williams, 1992) can be used for direct policy search by making weight adjustments in a direction that lies along the gradient of the expected reinforcement. Our architecture is used directly to do policy search, and its parameters are updated using REINFORCE. Let the attention network  $\psi$  be parametrized by  $\theta_u$  and  $\pi_R$  be parametrized by  $\theta_v$ . The updates are given by:

$$\theta_u \leftarrow \theta_u + \alpha_{\theta_u} (r - b) \frac{\partial \sum_{t=1}^M \log(\pi_T(s_t, a_t))}{\partial \theta_u} \quad (3)$$

$$\theta_v \leftarrow \theta_v + \alpha_{\theta_v} (r - b) \frac{\partial \sum_{t=1}^M \log(\pi_R(s_t, a_t))}{\partial \theta_v} \quad (4)$$

where  $\alpha_{\theta_u}, \alpha_{\theta_v}$  are non-negative factors,  $a_t$  is the action taken by the agent at state  $s_t$  following  $\pi_T$ ,

and  $\pi_R(s_t, a_t)$  is the probability of action  $a_t$  given by  $\pi_R$ .

### Policy Transfer in Actor-Critic using $A^2T$ :

Actor-Critic methods (Konda and Tsitsiklis, 2000) are Temporal Difference (TD) methods that have two separate components, *viz.*, an *actor* and a *critic*. The actor proposes a policy whereas the critic estimates the value function to critique the actor's policy. The updates to the actor happens through *TD-error* which is the one step estimation error that helps in reinforcing an agent's behaviour. We use  $A^2T$  for the actor part of the Actor-Critic. The actor,  $A^2T$  is aware of all the previous learnt tasks and tries to use those solutions for its benefit. The critic evaluates the action selection from  $\pi_T$  on the basis of the performance on the target task. With the same notations as REINFORCE for  $s_t, a_t, \theta_u, \theta_v, \alpha_{\theta_u}, \alpha_{\theta_v}, \pi_R, \pi_T$ ; let action  $a_t$  dictated by  $\pi_T$  lead the agent to next state  $s_{t+1}$  with a reward of  $r_{t+1}$  and let  $V(s_t)$  represent the value of state  $s_t$  and  $\gamma$  the discount factor. Then, the update equations are as below:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \quad (5)$$

$$\theta_u \leftarrow \theta_u + \alpha_{\theta_u} \delta_t \frac{\frac{\partial \log \pi_T(s_t, a_t)}{\partial \theta_u}}{\left| \frac{\partial \log \pi_T(s_t, a_t)}{\partial \theta_u} \right|} \quad (6)$$

$$\theta_v \leftarrow \theta_v + \alpha_{\theta_v} \delta_t \frac{\frac{\partial \log \pi_R(s_t, a_t)}{\partial \theta_v}}{\left| \frac{\partial \log \pi_R(s_t, a_t)}{\partial \theta_v} \right|} \quad (7)$$

### Value Transfer

In this case, the solutions being transferred are the source tasks' action-value functions, which we will call as  $Q$  functions. Thus,  $K_1, \dots, K_N, K_R \leftarrow Q_1, \dots, Q_N, Q_R$ .

### Value Transfer in Q learning using $A^2T$ :

Q-learning (Watkins and Dayan, 1992) is an off-policy Temporal Difference (TD) learning algorithm which learns state-action value  $Q$  function to guide the agent to selecting optimal action  $a$  at a state  $s$ .  $Q(s, a)$  is a measure of the long-term reward obtained by taking action  $a$  at state  $s$ . The Deep Q Network (DQN) (Mnih et al., 2015) approximates

the Q-value function with a deep neural network to be able to predict  $Q(s, a)$  over all actions  $a$ , for all states  $s$ , allowing its application in high dimensional state spaces. To avoid correlated updates from learning on the same transitions that the current network simulates, an experience replay (Lin, 1993)  $D$  (of fixed maximum capacity) is used, where the experiences are pooled in a FIFO fashion. We use DQN to learn our experts  $Q_i$  on the source tasks. Q-learning is used to ensure  $Q_T(s)$  is driven to a good estimate of  $Q$  functions for the target task. Taking advantage of the off-policy nature of Q-learning, both  $Q_R$  and  $Q_T$  can be learned from the experiences gathered by an  $\epsilon$ -greedy behavioral policy based on  $Q_T$ . Let the attention network  $\psi$  that outputs  $w$  be parametrised by  $\theta_u$  and the random network outputting  $Q_R$  be parametrised by  $\theta_v$ . Let  $\theta_u^-$  and  $\theta_v^-$  represent the parameters of the respective target networks. The updates equations are:

$$y^{Q_T} = (r + \gamma \max_{a'} Q_T(s', a'; \theta_u^-, \theta_v^-)) \quad (8)$$

$$L^{Q_T}(\theta_u, \theta_v) = \mathbb{E}_{s,a,r,s'} [(y^{Q_T} - Q_T(s, a; \theta_u, \theta_v))^2] \quad (9)$$

$$L^{Q_R}(\theta_v) = \mathbb{E}_{s,a,r,s'} [(y^{Q_T} - Q_R(s, a; \theta_v))^2] \quad (10)$$

$$\nabla_{\theta_u} L^{Q_T} = \mathbb{E}[(y^{Q_T} - Q_T(s, a)) \nabla_{\theta_u} Q_T(s, a)] \quad (11)$$

$$\nabla_{\theta_v} L^{Q_R} = \mathbb{E}[(y^{Q_T} - Q_R(s, a)) \nabla_{\theta_v} Q_R(s, a)] \quad (12)$$

$\theta_u$  and  $\theta_v$  are updated with the above gradients using RMSProp. The Q-learning updates on  $Q_R$  use the target value generated by  $Q_T$ . We use *target* networks for both  $Q_R$  and  $Q_T$  to stabilize the updates and reduce the non-stationarity as in DQN training. The parameters of the *target* networks are periodically updated to that of the *online* networks. Note that the usage of *target* here is to signify the parameters ( $\theta_u^-$ ,  $\theta_v^-$ ) used to calculate the *target* value in the Q-learning update and is different from its usage in the context of the *target* task.

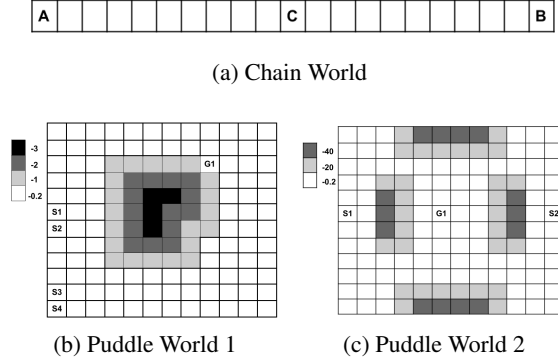


Figure 2: Different worlds used for our policy transfer experiments

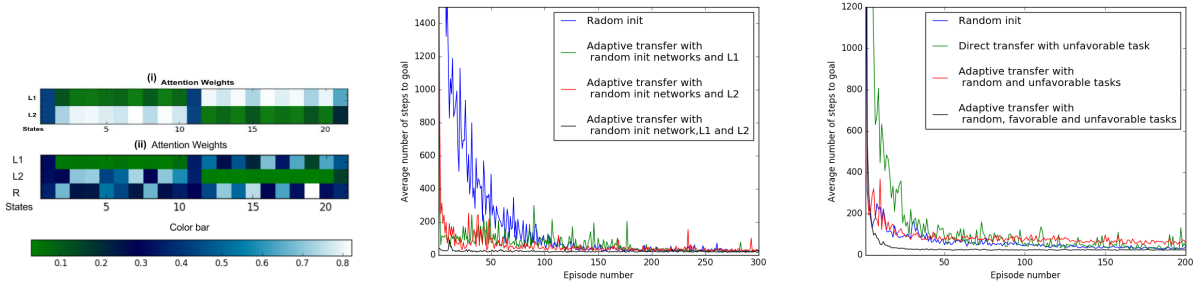
## Experiments and Discussion

We evaluate the performance of our architecture on policy transfer using two simulated worlds, *viz.*, chain world and puddle world as described below. One main motivation of these experiments is to test the consistency of results with the algorithm motivation.

**Chain world:** Figure 2a shows the chain world where the goal of the agent is to go from one point in the chain (starting state) to another point (goal state) in the least number of steps. At each state the agent can choose to either move one position to the left or to the right. After reaching the goal state the agent gets a reward that is inversely proportional to the number of steps taken to reach the goal.

**Puddle worlds:** Figures 2b and 2c show the discrete version of the standard puddle world that is widely used in Reinforcement Learning literature. In this world, the goal of the agent is to go from a specified start position to the goal position, maximising its return. At each state the agent can choose one of these four actions: move one position to the north, south, east or west. On reaching the goal state, the agent gets a reward of +10. On reaching other parts of the grid the agent gets different penalties as mentioned in the legend of the figures.

**Atari 2600:** We evaluate the performance of our architecture on value transfer using the Arcade Learning Environment (ALE) platform (Bellemare et al., 2012). ALE provides a simulator for Atari 2600 games. This is one of the most commonly used benchmark tasks for deep reinforcement learning algorithms (Mnih et al., 2015), (Mnih et al., 2016),



(a) The weights given by the attention network (b) Transferring from multiple similar source tasks (c) Avoiding negative transfer and transferring from a favorable task

Figure 3: Results of the policy transfer experiments

(Parisotto et al., 2015), (Rusu et al., 2016). We perform our adaptive transfer learning experiments on the Atari 2600 game Pong.

### Ability to do Selective Transfer

In this section, we consider the case when multiple partially favorable source tasks are available such that each of them can assist the learning process for different parts of the state space of the target task. The objective here is to first show the effectiveness of the attention network in learning to *focus* only on the source task relevant to the state the agent encounters while trying to complete the target task and then evaluating the full architecture with an additional randomly initialised network.

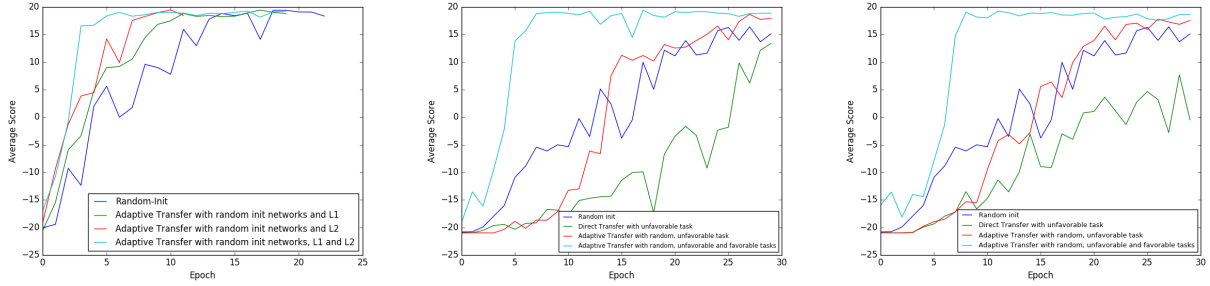
This is illustrated for the Policy Transfer setting using the chain world shown in (Fig. 2a). Consider that the target task  $LT$  is to start in  $A$  or  $B$  with uniform probability and reach  $C$  in the least number of steps. Now, consider that two learned source tasks, *viz.*,  $L1$  and  $L2$ , are available.  $L1$  is the source task where the agent has learned to reach the left end ( $A$ ) starting from the right end ( $B$ ). In contrast,  $L2$  is the source task where the agent has learned to reach the right end ( $B$ ) starting from the left end ( $A$ ). Intuitively, it is clear that the target task should benefit from the policies learnt for tasks  $L1$  and  $L2$ . We learn to solve the task  $LT$  using REINFORCE given the policies learned for  $L1$  and  $L2$ . Figure 3a (i) shows the weights given by the attention network to the two source task policies for different parts of the state space at the end of learning. We observe that the attention network has learned to ignore  $L1$ , and  $L2$  for the left, and right half of the state space of the

target task, respectively.

Next, we add the randomly initialised network and evaluate the full architecture on this task. Figure 3a (ii) shows the weights given by the attention network to the different source policies for different parts of the state space at the end of learning. We observe that the attention network has learned to ignore  $L1$ , and  $L2$  for the left, and right half of the state space of the target task, respectively. As the randomly initialised actor network outputs better policies over time, it has a high weight throughout the state space of the target task.

We also evaluate our architecture in a relatively more complex puddle world shown in Figure 2c. In this case,  $L1$  is the task of moving from  $S1$  to  $G1$ , and  $L2$  is the task of moving from  $S2$  to  $G1$ . In the target task  $LT$ , the agent has to learn to move to  $G1$  starting from either  $S1$  or  $S2$  chosen with uniform probability. We learn the task  $LT$  using Actor-Critic method, where the following are available (i) learned policy networks for  $L1$  (ii) learned policy network for  $L2$  and (iii) a randomly initialized policy network. Figure 3b shows the performance results. We observe that actor-critic using  $A^2T$  is able to use the policies learned for  $L1$ , and  $L2$  and performs better than a network learning from scratch without any knowledge of source tasks.

We do a similar evaluation of the attention network, followed by our full architecture for value transfer as well. We create partially useful source tasks through a modification of the Atari 2600 game Pong. We take inspiration from a real world scenario in the sport Tennis, where one could imagine two different right-handed (or left) players with



(a) Transferring from multiple different source tasks (b) Avoiding negative transfer (Pong) and transferring from a favorable task (c) Avoiding negative transfer (Freeway) and transferring from a favorable task

Figure 4: Value transfer experiments results: A training epoch is 250,000 frames and for each training epoch, we evaluate the networks with a testing epoch that lasts 125,000 frames. We report the average score over the completed episodes for each testing epoch. The average scores obtained this way are averaged over 2 runs with different random seeds. In the testing epochs, we use  $\epsilon = 0.05$  in the  $\epsilon$ -greedy policy. More specific training and architecture details mentioned in APPENDIX.

the first being an expert player on the forehand but weak on the backhand, while the second is an expert player on the backhand but weak on the forehand. For someone who is learning to play tennis with the same style (right/left) as the experts, it is easy to follow the forehand expert player whenever he receives a ball on the forehand and follow the backhand expert whenever he receives a ball on the backhand.

We try to simulate this scenario in Pong. The trick is to blur the part of the screen where we want to force the agent to be weak at returning the ball. The blurring we use is to just black out all pixels in the specific region required. To make sure the blurring doesn't contrast with the background, we modify Pong to be played with a black background (pixel value 0) instead of the existing gray (pixel value 87). We construct two partially helpful source task experts  $L1$  and  $L2$ .  $L1$  is constructed by training a DQN on Pong with the upper quadrant (the agent's side) blurred, while  $L2$  is constructed by training a DQN with the lower quadrant (the agent's side) blurred. This essentially results in the ball being invisible when it is in the upper quadrant for  $L1$  and lower quadrant for  $L2$ . We therefore expect  $L1$  to be useful in guiding to return balls on the lower quadrant, and  $L2$  for the upper quadrant. The goal of the attention network is to learn suitable filters and parameters so that it will focus on the correct source task for a specific situation in the game. The source task experts  $L1$  and  $L2$  scored an average (over

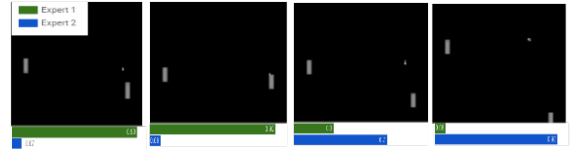


Figure 5: Visualisation of the attention weights in the Selective Transfer with Attention Network experiment: Green and Blue bars signify the attention probabilities for Expert-1 ( $L1$ ) and Expert-2 ( $L2$ ) respectively. We see that in the first two snapshots, the ball is in the lower quadrant and as expected, the attention is high on Expert-1, while in the third and fourth snapshots, as the ball bounces back into the upper quadrant, the attention increases on Expert-2.

125,000 frames) of **9.2** and **8** respectively on Pong game play with black background. With an attention network to suitably weigh the value functions of  $L1$  and  $L2$ , an average performance of **17.2** was recorded just after a single epoch (250,000 frames) of training. (The score in Pong is in the range of  $[-21, 21]$ ). This clearly shows that the attention mechanism has learned to take advantage of the experts adaptively. Fig. 5 shows a visualisation of the attention weights for the same. We then evaluate our full architecture ( $A^2T$ ) in this setting, i.e with an addition of DQN learning from scratch to the above setting. The architecture can take advantage of the knowledge of the source task experts selectively early on during the training while improving

the expertise of the randomly initialized target network to perform well on the target task. Figure 4a summarizes the results, where it is clear that learning with both the partially useful experts is better than learning with only one of them which in turn is better than learning from scratch without any additional knowledge.

### Ability to Avoid Negative Transfer and Ability to Transfer from Favorable Task

We first consider the case when only one learned source task is available such that its solution  $K_1$  (policy or value) can hamper the learning process of the new target task. We refer to such a source task as an unfavorable source task. In such a scenario, the attention network shown in Figure 1 should learn to assign a very low weight (ignore) to  $K_1$ . We also consider a modification of this setting by adding another source task whose solution  $K_2$  is favorable to the target task. In such a scenario, the attention network should learn to assign high weight (attend) to  $K_2$  while ignoring  $K_1$ .

We now define an experiment using the puddle world from Figure 2b for policy transfer. The target task in our experiment is to maximize the return in reaching the goal state  $G1$  starting from any one of the states  $S1, S2, S3, S4$ . We artificially construct an unfavorable source task by first learning to solve the above task and then negating the weights of the topmost layer of the actor network. We then add a favorable task to the above setting. In such a scenario, the attention network should learn to assign high weights to the policy output of the favorable source task, while giving low weights to that of the negative source task. We artificially construct a favorable source task simply by learning to solve the target task and using the learned actor network. Figure 3c shows the results.

The target task for the value transfer experiment is to reach expert level performance on Pong. We construct two kinds of unfavorable source tasks for this experiment.

**Inverse-Pong:** A DQN on Pong trained with negated reward functions, that is with  $R'(s, a) = -R(s, a)$  where  $R(s, a)$  is the reward provided by the ALE emulator for choosing action  $a$  at state  $s$ .

**Freeway:** An expert DQN on another Atari 2600 game, Freeway, which has the same range of optimal

value functions and same action space as Pong. We empirically verified that the Freeway expert DQN leads to negative transfer when directly initialized and fine-tuned on Pong which makes this a good proxy for a negative source task expert even though the target task Pong has a different state space.

We artificially construct a favorable source task by learning a DQN to achieve expertise on the target task (Pong) and use the learned network. Figure 4b compares the performance of the various scenarios when the unfavorable source task is Inverse-Pong, while Figure 4c offers a similar comparison with the negative expert being Freeway.

From all the above results, we can clearly see that  $A^2T$  does not get hampered by the unfavorable source task by learning to ignore the same and performs competitively with just a randomly initialized learning on the target task without any expert available. Secondly, in the presence of an additional source task that is favorable,  $A^2T$  learns to transfer useful knowledge from the same while ignoring the unfavorable task, thereby reaching expertise on the target task much faster than the other scenarios.

### Conclusion and Future work

In this paper we present a very general deep neural network architecture,  $A^2T$  for transfer learning that avoids negative transfer while enabling selective transfer from multiple source tasks. We show simple ways of using  $A^2T$  for policy transfer and value transfer. We empirically evaluate its performance with different algorithms, using simulated worlds and games, and show that it indeed achieves its stated goals. Apart from transferring task solutions,  $A^2T$  can also be used for transferring other useful knowledge such as the model.

While in this work we focused on transfer between tasks that share the same state and action spaces, the use of deep networks opens up the possibility of going beyond this setting. For example, a deep neural network can be used to learn common representations (Parisotto et al., 2015) for multiple tasks thereby enabling transfer between related tasks that could possibly have different state-action spaces. A hierarchical attention over the lower level filters across source task networks while learning the filters for the target task network is another natu-



ral extension to transfer across tasks with different state-action spaces. We would also like to explore this setting in avoiding negative transfer in continuous control tasks since negative transfer has practical importance in Robotics. Additionally, we believe that this framework is an important tool to build upon and apply for source-domain adaptation, where we could selectively decide to pick information of the source domain (model) in addition to using the solutions of the source tasks while learning to solve the target task. This way, model based approaches could take advantage of the model information from the source domains in addition to the source task solutions (policies or value functions). Over all, we believe that  $A^2T$  is a novel way to approach transfer learning that opens up many new avenues of research in this area.

## Acknowledgements

We would like to thank Charu Chauhan, Sarath Chandar, Yoshua Bengio, Caglar Gulchere for useful feedback about the work.

## References

- Christopher G Atkeson and Stefan Schaal. 1997. Robot learning from demonstration. In *Proceedings of International Conference on Machine Learning*, volume 97.
- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- Bikramjit Banerjee and Peter Stone. 2007. General game learning using knowledge transfer. In *The 20th International Joint Conference on Artificial Intelligence*.
- Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2012. The arcade learning environment: An evaluation platform for general agents. *arXiv preprint arXiv:1207.4708*.
- Emma Brunskill and Lihong Li. 2014. Pac-inspired option discovery in lifelong reinforcement learning. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 316–324.
- Kimberly Ferguson and Sridhar Mahadevan. 2006. Proto-transfer learning in markov decision processes using spectral methods. *Computer Science Department Faculty Publication Series*, page 151.
- Fernando Fernández and Manuela Veloso. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727. ACM.
- Vijay Konda and John Tsitsiklis. 2000. Actor-critic algorithms. In *SIAM Journal on Control and Optimization*, pages 1008–1014. MIT Press.
- George Konidaris, Ilya Scheidwasser, and Andrew G Barto. 2012. Transfer in reinforcement learning via shared features. *The Journal of Machine Learning Research*, 13(1):1333–1371.
- Alessandro Lazaric and Marcello Restelli. 2011. Transfer from multiple mdps. In *Advances in Neural Information Processing Systems*, pages 1746–1754.
- Long-Ji Lin. 1993. Reinforcement learning for robots using neural networks. Technical report, DTIC Document.
- Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. 2004. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM.
- Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.
- Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. *arXiv preprint arXiv:1602.01783*.
- Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. 2013. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9.
- Emilio Parisotto, Jimmy Ba, and Ruslan Salakhutdinov. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342.
- Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. *CoRR*, abs/1606.04671.
- Jonathan Sorg and Satinder Singh. 2009. Transfer via soft homomorphisms. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 741–748. International Foundation for Autonomous Agents and Multiagent Systems.

Ivan Sorokin, Alexey Seleznev, Mikhail Pavlov, Aleksandr Fedorov, and Anastasiia Ignateva. 2015. Deep attention recurrent q-network. *arXiv preprint arXiv:1512.01693*.

Erik Talvitie and Satinder Singh. 2007. An experts algorithm for transfer learning. In *Proceedings of the 20th international joint conference on Artificial intelligence*, pages 1065–1070. Morgan Kaufmann Publishers Inc.

Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685.

Matthew E Taylor and Peter Stone. 2011. An introduction to intertask transfer for reinforcement learning. *AI Magazine*, 32(1):15.

Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning*, 8(3):279–292.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

## APPENDIX A: Details of the Network Architecture in Value Transfer Experiments

For the source task expert DQNs, we use the same architecture as (Mnih et al., 2015) where the input is  $84 \times 84 \times 4$  with 32 convolution filters, dimensions  $8 \times 8$ , stride  $4 \times 4$  followed by 64 convolution filters with dimensions  $4 \times 4$  and stride  $2 \times 2$ , again followed by 64 convolution filters of size  $3 \times 3$  and stride  $1 \times 1$ . This is then followed by a fully connected layer of 512 units and finally by a fully connected output layer with as many units as the number of actions in Pong (Freeway) which is 3. We use ReLU nonlinearity in all the hidden layers.

For the random initialized network on the target task, we retain the same architecture as above. For the attention network, we retain all of the architecture as above except for the output layer, which has a softmax nonlinearity and as many units as the number of tasks we consider. If there are  $N$  source tasks and 1 random network, we would have an output layer with  $N + 1$  units. If the experiment is to show the performance of only the attention network, we would have as many units as the number of source tasks.

## APPENDIX B: Training Details

### Training Algorithm

For all our experiments in Value Transfer, we used RMSProp as in (Mnih et al., 2015) for our gradient updates. For Policy Transfer, since the tasks were simple, stochastic gradient descent was sufficient to provide stable updates. We also use reward clipping, target networks and experience replay for our value transfer experiments in exactly the same way (all hyper parameters retained) as (Mnih et al., 2015).

### Learning Rate

We run all experiments with two learning rates, 0.0025 and 0.0005. In general, the lower learning rate provided more stable (less variance) training curves. While comparing across algorithms, we picked the best performing learning rate out of the two (0.0025 and 0.0005) for each training curve.

## APPENDIX C: Blurring Experiments on Pong

The experts are trained with blurring (hiding the ball) and black background as illustrated in APPENDIX A. Therefore, to compare the learning with that of a random network without any additional knowledge, we ran the baseline DQN on Pong with a black background too. Having a black background provides a rich contrast between the white ball and the black background, thereby making training easier and faster, which is why the performance curves in that setting are different to the other two settings reported for Inverse Pong and Freeway Negative transfer experiments where no blacking is done and Pong is played with a gray background. The blurring mechanism in Pong is illustrated in APPENDIX E.

## APPENDIX D: Blurring experiments on Breakout

Similar to our Blurring experiment on Pong, we additionally ran another experiment on the Atari 2600 game Breakout to validate the efficiency of our attention mechanism. We consider a setup with two experts  $L1$  and  $L2$  along with our attention network. The experts  $L1$  and  $L2$  were trained by blurring the lower left and right quadrants of the breakout screen respectively. We don’t have to make the background

## APPENDIX E: Blurring Mechanism in Pong - Details

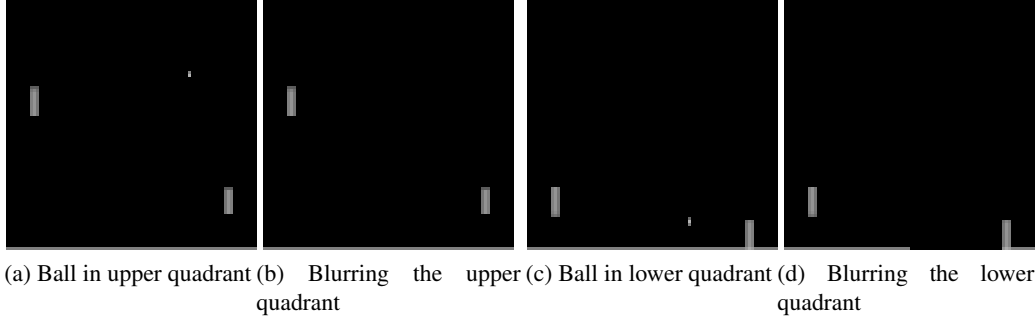


Figure 6: The figures above explain the blurring mechanism for selective transfer experiments on Pong. The background of the screen is made black. Let  $X$  ( $84 \times 84$ ) denote an array containing the pixels of the screen. The paddle controlled by the agent is the one on the right. We focus on the two quadrants  $X1 = X[: 42, 42 :]$  and  $X2 = X[42 :, 42 :]$  of the Pong screen relevant to the agent controlled paddle. To simulate an expert that is weak at returning balls in the upper quadrant, the portion of  $X1$  till the horizontal location of agent-paddle, ie  $X1[:, : 31]$  is blacked out, while similarly, for simulating weakness in the bottom quadrant, we blur the portion of  $X2$  till the agent-paddle’s horizontal location, ie  $X2[:, : 31] = 0$ . Figures 6a and 6b illustrate the scenarios of blurring the upper quadrant before and after blurring; and similarly do 6c and 6d for blurring the lower quadrant. Effectively, blurring this way with a black screen is equivalent to hiding the ball (white pixel) in the appropriate quadrant where weakness is to be simulated. Hence, Figures 6b and 6d are the mechanisms used while training a DQN on Pong to hide the ball at the respective quadrants, so to create the partially useful experts which are analogous to forehand-backhand experts in Tennis.  $X[: a, : b]$  indicates the subarray of  $X$  with all rows upto row index  $a$  and all columns upto column index  $b$ .

black like in the case of Pong because the background is already black in Breakout and direct blurring is sufficient to hiding the ball in the respective regions without any contrasts introduced. We blur only the lower part so as to make it easy for the agent to at least anticipate the ball based on the movement at the top. We empirically observed that blurring the top half (as well) makes it hard to learn any meaningful partially useful experts  $L1$  and  $L2$ .

The goal of this experiment is to show that the attention network can learn suitable filters so as to dynamically adapt and learn to select the expert appropriate to the situation (game screen) in the task. The expert  $L1$  which was blurred on the left bottom half is bound to weak at returning balls on that region while  $L2$  is expected to be weak on the right. This is in the same vein as the forehand-backhand example in Tennis and its synthetic simulation for Pong by blurring the upper and lower quadrants. During game play, the attention mechanism is expected to ignore  $L2$  when the ball is on the bottom right half (while focusing on  $L1$ ) and similarly ignore  $L2$

(while focusing on  $L1$ ) when the ball is on the left bottom half. We learn experts  $L1$  and  $L2$  which score **42.2** and **39.8** respectively. Using the attention mechanism to select the correct expert, we were able to achieve a score of **94.5** after training for 5 epochs. Each training epoch corresponds to 250,000 decision steps, while the scores are averaged over completed episodes run for 125,000 decision steps. This shows that the attention mechanism learns to select the suitable expert. Though the performance is limited by the weaknesses of the respective experts, our goal is to show that the attention paradigm is able to take advantage of both experts appropriately. This is evident from the scores achieved by standalone experts and the attention mechanism. Additionally, we also present a visualization of the attention mechanism weights assigned to the experts  $L1$  and  $L2$  during game play in APPENDIX G. The weights assigned are in agreement with what we expect in terms of selective attention. The blurring mechanism is visually illustrated in APPENDIX F.

## APPENDIX F: Blurring Mechanism in Breakout - Details

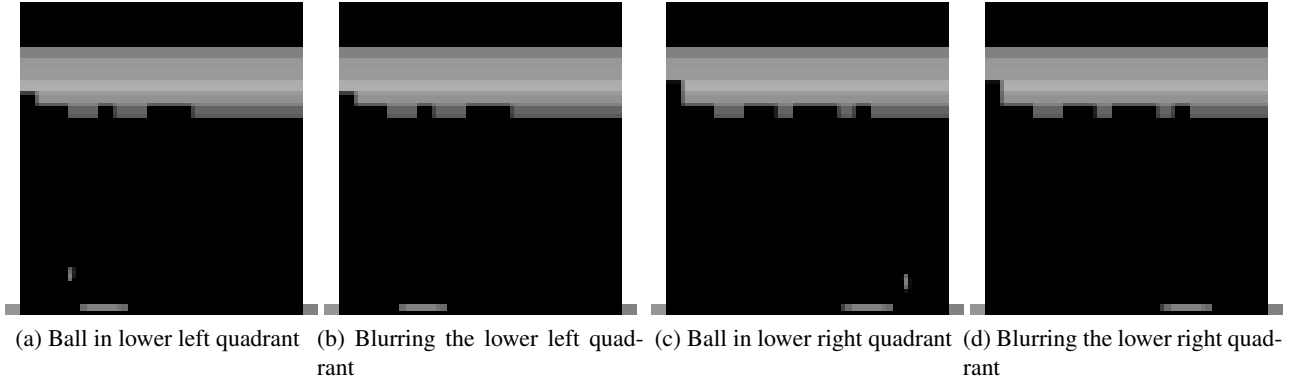


Figure 7: The figures above explain the blurring mechanism used for selective transfer experiments on Breakout. The background the screen is already black. Let  $X$  ( $84 \times 84$ ) denote an array containing the pixels of the screen. The paddle controlled by the agent is the one on the right. We focus on the two quadrants  $X1 = X[31 : 81, 4 : 42]$  and  $X2 = X[31 : 81, 42 : 80]$ . We perform blurring in each case by ensuring  $X1 = 0$  and  $X2 = 0$  for all pixels within them for training  $L1$  and  $L2$  respectively. Effectively, this is equivalent to hiding the ball in the appropriate quadrants. Blurring  $X1$  simulates weakness in the lower left quadrant, while blurring  $X2$  simulates weakness in the lower right quadrant. We don't blur all the way down upto the last row to ensure the paddle controlled by the agent is visible on the screen. We also don't black the rectangular border with a width of 4 pixels surrounding the screen. Figures 7a and 7b illustrate the scenarios of blurring the lower left quadrant before and after blurring; and similarly do 7c and 7d for blurring the lower right quadrant.

## APPENDIX G: Blurring Attention Visualization on Breakout

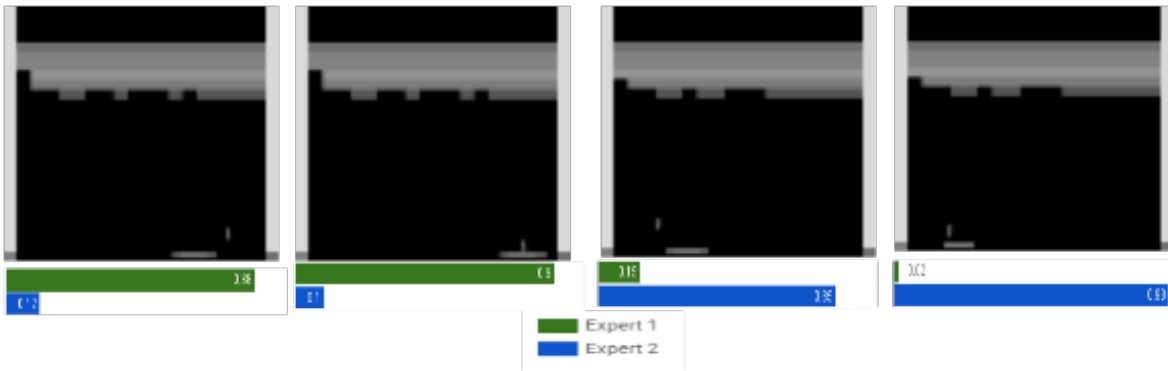


Figure 8: Visualisation of the attention weights in the Selective Transfer with Attention for Breakout: Green and Blue bars signify the attention probabilities for Expert-1 ( $L1$ ) and Expert-2 ( $L2$ ) respectively on a scale of  $[0, 1]$ . We see that in the first two snapshots, the ball is in the lower right quadrant and as expected, the attention is high on Expert-1, while in the third and fourth snapshots, the ball is in the lower left quadrant and hence the attention is high on Expert-2.